HW

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/037,666	01/03/2002	Sriram Vajapeyam	42390P11927	8277

8791 7590 03/07/2005

BLAKELY SOKOLOFF TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1030

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 03/07/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/037,666	VAJAPEYAM ET AL.	
	Examiner	Art Unit	
	David J. Huisman	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *28 December 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-30* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-30* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *28 December 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-30 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Amendment as received on 12/28/2004.

Specification

3. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed. MPEP §606.01 states "This may result in slightly longer titles, but the loss in brevity of title will be more than offset by the gain in its informative value in indexing, classifying, searching, etc. If a satisfactory title is not supplied by the applicant, the examiner may, at the time of allowance, change the title by examiner's amendment."

4. The disclosure is objected to because of the following informalities: In the last line of amended paragraph [0025], please replace "issued" with --issue--.

Appropriate correction is required.

Claim Objections

5. Claim 8 is objected to because of the following informalities: Please replace "description" with --descriptor--. Appropriate correction is required.

6. Claim 9 is objected to because of the following informalities: Please replace the phrase "a plurality of dependency descriptor" with --a plurality of dependency descriptors-- (note plural descriptors). Also, replace "description" with --descriptor--. Appropriate correction is required.

7. Claim 11 is objected to because of the following informalities: Please replace "dependency claim" with --dependency data--. Appropriate correction is required.

8. Claim 14 is objected to because of the following informalities: It is not clear why applicant deleted the word "store" at the end of the claim. For purposes of this examination, the examiner will assume that "store" should exist. Appropriate correction is required.

9. Claim 17 is objected to because of the following informalities: Please replace "description" with --descriptor--. Appropriate correction is required.

10. Claim 18 is objected to because of the following informalities: Please replace "description" with --descriptor--. Appropriate correction is required.

Claim Rejections - 35 USC § 112

11. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

12. Claim 28 recites the limitation "the dependencies" in the last two lines of the claim. There is insufficient antecedent basis for this limitation in the claim.

Maintained Rejections

13. Applicant has failed to overcome the prior art rejections for claims 20-27 set forth in the previous Office Action. Consequently, these rejections are respectfully maintained by the

examiner and are copied below for applicant's convenience. It should be noted that the wording of the claim rejection has been modified to account for applicant's amendments to the claims.

Withdrawn Rejections

14. Applicant has overcome the rejection of claims 1-19 and 28-30 set forth in the previous Office Action. Consequently, these rejections are hereby withdrawn by the examiner any associated arguments are moot. However, upon further consideration, a new ground(s) of rejection is applied below.

Claim Rejections - 35 USC § 102

15. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

16. Claims 1, 10-19 and 28-30 are rejected under 35 U.S.C. 102(b) as being anticipated by Ranganathan et al., "The PEWs Microarchitecture: Reducing Complexity Through Data Dependence-Based Decentralization," 1998 (as disclosed by applicant and herein referred to as Ranganathan).

17. Referring to claim 1, Ranganathan has taught a logic circuit comprising:

a) a control flow logic to select a trace descriptor for processing including at least one dependency descriptor, the dependency descriptor including dependency information for an instruction sequence and a location of the instruction sequence. See the first paragraph of section

3.3 and note that data dependence information (dependency descriptor) associated with a particular trace of instructions is stored in the trace cache. The trace descriptor would include all information within the trace cache associated with a particular trace of instructions (associated dependency information, generated bitmaps (section 3.2.2), and any other related information). In addition, the dependency descriptor includes information about dependencies, which are inherently between source and destination registers of instructions. Therefore, register information must be indicated by the descriptor. And, registers, in turn are locations of the instruction sequence.

b) Ranganathan has taught that an instruction sequence itself is stored in the trace cache. See the last paragraph of section 3.1. Ranganathan has not taught that the dependency descriptor includes a location (i.e., address) of the instruction sequence. However, Nair has taught a component called a Superblock Target Buffer (STB), which acts like a common trace cache in that it tries to predict long execution paths. However, unlike a common trace cache, the STB stores instruction addresses and not the instructions themselves. See column 10, lines 11-17. Verbauwhede has shown that instruction addresses may be smaller than the instructions themselves. See column 2, lines 43-49. More specifically, Verbauwhede shows an example where an instruction address (location) may be specified by 16 bits where the instructions themselves are 32 bits. A person of ordinary skill in the art would have recognized that by storing the location of the instruction sequence as opposed to the instruction sequence itself, less space would be consumed in the trace cache. In addition, it should be realized that it is not necessarily important whether instructions or instruction addresses (locations) are stored in a trace cache. Instead, all that is required is that instructions may be identified by the trace cache

and this may be done by providing the instruction or its address. As a result, it would have been obvious to one or ordinary skill in the art at the time of the invention to modify Ranganathan such that the trace cache (more specifically, the dependency descriptor) stores an instruction location as opposed to the instruction sequence itself.

b) a data flow logic coupled to the control flow logic to execute the instruction sequence according to the dependency information stored in the dependency descriptor. See the abstract.

18. Referring to claim 10, Ranganathan has taught a computer system comprising:

a) at least one memory device. See the data cache banks of Fig.2.

b) a bus coupled to the at least one memory device. It is inherent a bus is connected to the memory device so data is able to travel between memory and the rest of the system.

c) a control flow logic device to analyze dependencies among instruction sequences in a trace and create a trace descriptor comprising a dependency descriptor indicating a live-in value for an instruction sequence. See Fig.3, the abstract, and section 3.3. Note that dependency info is determined, stored in the trace cache, and used for placing instructions in certain PEWs. The dependency info makes up the dependency descriptor and it indicates live-in values, which are values required by instructions of the associated trace. For instance, see Fig.5, and note the register dependencies, which must be specified by the dependency descriptor. These registers represent live in values upon which instructions depend.

d) a data flow logic device coupled to the control flow logic to execute a plurality of the instruction sequences according to the dependency information stored in the dependency descriptor. See Fig.3 and the abstract.

19. Referring to claim 11, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught a dependency claim issue window coupled between the control flow logic device and the data flow logic device, the dependency data issue window to store the dependency descriptor. See section 3.3 and Fig.3, and note the usher component. The dependency information is sent (and inherently stored) within the usher so that it may determine how to place the instructions.

20. Referring to claim 12, Ranganathan has taught a computer system as described in claim 11. Ranganathan has further taught a second storage area coupled to the control flow logic device, the second storage area dedicated to storage of trace descriptors. See Fig.3 (trace cache).

21. Referring to claim 13, Ranganathan has taught a computer system as described in claim 12. Ranganathan has further taught a third storage area coupled to the data flow logic device, the third storage area to store instructions contiguously based on dependency information. See Fig.3 (the PEW queues). Instructions are sent to the PEWs based on dependency information and each would be stored until its turn to execute.

22. Referring to claim 14, Ranganathan has taught a computer system as described in claim 13. Ranganathan has further taught a fourth storage area coupled to the data flow logic device and control flow logic device, the fourth storage area to store live-out data. See Fig.2, and note the ISA-visible register file. Clearly, instructions of a particular group (for instance, those of Fig.5) will be writing to registers within the file. These registers may in turn be used by a subsequent group of instructions.

23. Referring to claim 15, Ranganathan has taught a computer system as described in claim 14. Ranganathan has further taught a fifth storage area coupled to the control flow logic, the

fifth storage area to map live-in and live-out data. See Fig.3, and note the register queues. The register queues, according to section 3.4, implement register-renaming, which is known to be the mapping of live-in and live-out data.

24. Referring to claim 16, Ranganathan has taught a computer system as described in claim 15. Ranganathan has further taught that each of the storage areas is in at least one memory device. This is inherent because any component which includes a storage area is a memory device.

25. Referring to claim 17, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught that the trace descriptor includes aggregate live-in data for a plurality of dependency descriptors in the trace description. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions depend on data which was produced prior to execution of the given trace. This data is live-in data and is associated with registers, for instance. Clearly, the trace descriptor will include this live-in data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-in data is for a plurality of dependency descriptors.

26. Referring to claim 18, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught that the trace descriptor includes aggregate live-out data for a plurality of dependency descriptors in the trace description. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions produce data which will be depended upon in the future. This data is live-out data and is associated with registers, for instance.

Clearly, the trace descriptor will include this live-out data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-out data is for a plurality of dependency descriptors.

27. Referring to claim 19, Ranganathan has taught a computer system as described in claim 10. Ranganathan has further taught that the dependency descriptor includes a live-out value for the instruction sequence. As previously discussed, the dependency descriptor includes data which specifies data dependencies, which in turn means that the data is associated with registers (which represent live-in and live-out values).

28. Referring to claim 28, Ranganathan has taught a machine-readable medium that provides instructions, which when executed by a machine cause the machine to perform operations comprising:

a) fetching a trace descriptor and separating out a dependency descriptor including dependency information for a set of instructions from the trace descriptor. See the first paragraph of section 3.3 and note that data dependence information (dependency descriptor) associated with a particular trace of instructions is stored in the trace cache. The trace descriptor would include all information within the trace cache associated with a particular trace of instructions (associated dependency information, generated bitmaps (section 3.2.2), and any other related information).

b) storing the dependency descriptor in an issue window to await assignment to an execution unit. See section 3.3 and Fig.3, and note the usher component. The dependency information is

sent (and inherently stored) within the usher so that it may determine how to place the instructions.

c) fetching the set of instructions described by dependency descriptor. Since instructions are sent to appropriate execution units based on the dependency descriptor, they must be fetched.

d) executing a plurality of the instruction sequences according to the dependencies stored in the dependency descriptor. See the abstract.

29. Referring to claim 29, Ranganathan has taught a medium as described in claim 28. Ranganathan has further taught that the operations further comprise updating live-out data in a first storage area. It is known that live-out data is data which is generated by a given trace (instruction sequence) that is needed by another trace for input. Instructions within a trace will inherently write to registers which hold the live-out data which may be read by another trace. For instance, see Fig.5.

30. Referring to claim 30, Ranganathan has taught a method as described in claim 29. Ranganathan has further taught reading the issue window into the data flow logic. Clearly, the dependency descriptor will be read from storage within the usher and used by "Data flow logic" to determine how to breakup and assign instructions to execution units.

31. Claims 20-22 are rejected under 35 U.S.C. 102(e) as being anticipated by Batten et al., U.S. Patent No. 6,260,189 (as applied in the previous Office Action and herein referred to as Batten).

32. Referring to claim 20, Batten has taught a method of processing instructions comprising:

a) fetching a trace descriptor. See Fig.9 and note the ccdd instruction (trace descriptor). Being an instruction, it is inherently fetched.

b) separating out a dependency descriptor including dependency information for a set of instructions from the trace descriptor. Note the format of the trace descriptor in Fig.8. The dtype field and numInstr field make up a dependency descriptor which specifies the types of dependencies found within a following instruction sequence. The descriptor is separated out when the system needs to determine what type of dependencies exist.

c) fetching the set of instructions at a location indicated by the dependency descriptor. Note from Fig.9, for instance, that each of the instructions in the instruction sequence will inherently be fetched before they are executed. Also, the dependency descriptor will specify that the next X number of instructions at the next X number of addresses will have the associated dependencies. This is specified by the numInstr field shown in Fig.7. The location of the sequence itself is not explicitly stored in the dependency descriptor, but the location of the sequence is indicated by knowing how many immediately subsequent instructions are described by the dependency descriptor.

d) executing a plurality of the instruction sequences according to the dependencies stored in the dependency descriptor. See column 6, lines 44-58, and column 3, lines 52-57. Note that the dependency information is used by the hardware in executing the instructions such that stalls may be avoided by disabling the components which check for stalls.

33. Referring to claim 21, Batten has taught a method as described in claim 20. Batten has further taught updating live-out data in a first storage area. It is known that live-out data is data

which is generated by a given trace (instruction sequence) that is needed by another trace for input. Looking at Fig.9-10, for instance, assume that the trace of Fig.10 follows the trace of Fig.9. It can be seen that r9, s5, and s3, of Fig.9 are live-out values because they are generated by the trace of Fig.9 and they are used as inputs by the trace of Fig.10. And, clearly these values are stored within registers, which are a fourth storage.

34. Referring to claim 22, Batten has taught a method as described in claim 21. Batten has further taught:

a) storing the dependency descriptor extracted by the control flow logic into a second storage area. See Fig.7, and note that the dependency descriptor, which is ultimately extracted, is stored in the memory storage associated with the dtype field.

b) reading the descriptor out of the second storage area into the data flow logic. Clearly, the information regarding the dependencies is read so that the hardware knows how to go about executing the associated instruction sequence.

Claim Rejections - 35 USC § 103

35. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

36. Claims 1-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over Ranganathan, as applied above, in view of Nair, U.S. Patent No. 6,304,962. Furthermore, Verbauwhede, U.S.

Patent No. 5,732,255, is cited as extrinsic evidence for showing a reason why it would be more beneficial to store addresses in a trace cache as opposed to instructions.

37. Referring to claim 1, Ranganathan has taught a logic circuit comprising:

a) a control flow logic to select a trace descriptor for processing including at least one dependency descriptor, the dependency descriptor including dependency information for an instruction sequence. See the first paragraph of section 3.3 and note that data dependence information (dependency descriptor) associated with a particular trace of instructions is stored in the trace cache. The trace descriptor would include all information within the trace cache associated with a particular trace of instructions (associated dependency information, generated bitmaps (section 3.2.2), and any other related information).

b) Ranganathan has taught that an instruction sequence itself is stored in the trace cache. See the last paragraph of section 3.1. Ranganathan has not taught that the dependency descriptor includes a location (i.e., address) of the instruction sequence. However, Nair has taught a component called a Superblock Target Buffer (STB), which acts like a common trace cache in that it tries to predict long execution paths. However, unlike a common trace cache, the STB stores instruction addresses and not the instructions themselves. See column 10, lines 11-17. Verbauwhede has shown that instruction addresses may be smaller than the instructions themselves. See column 2, lines 43-49. More specifically, Verbauwhede shows an example where an instruction address (location) may be specified by 16 bits where the instructions themselves are 32 bits. A person of ordinary skill in the art would have recognized that by storing the location of the instruction sequence as opposed to the instruction sequence itself, less space would be consumed in the trace cache. In addition, it should be realized that it is not

necessarily important whether instructions or instruction addresses (locations) are stored in a trace cache. Instead, all that is required is that instructions may be identified by the trace cache and this may be done by providing the instruction or its address. As a result, it would have been obvious to one or ordinary skill in the art at the time of the invention to modify Ranganathan such that the trace cache (more specifically, the dependency descriptor) stores an instruction location as opposed to the instruction sequence itself.

b) a data flow logic coupled to the control flow logic to execute the instruction sequence according to the dependency information stored in the dependency descriptor. See the abstract.

38. Referring to claim 2, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught a first storage area coupled to the control flow logic and the data flow logic, the first storage area to store the dependency descriptor. See section 3.3 and Fig.3, and note the usher component. The dependency information is sent (and inherently stored) within the usher so that it may determine how to place the instructions.

39. Referring to claim 3, Ranganathan in view of Nair has taught a logic circuit as described in claim 2. Ranganathan has further taught a second storage area coupled to the control flow logic, the second storage area dedicated to storage of trace descriptors. See Fig.3 (trace cache).

40. Referring to claim 4, Ranganathan in view of Nair has taught a logic circuit as described in claim 3. Ranganathan has further taught a third storage area coupled to the data flow logic, the third storage area to store instructions contiguously based on dependency information. See Fig.3 (the PEW queues). Instructions are sent to the PEWs based on dependency information and each would be stored until its turn to execute.

41. Referring to claim 5, Ranganathan in view of Nair has taught a logic circuit as described in claim 4. Ranganathan has further taught a fourth storage area coupled to the data flow logic and control flow logic, the fourth storage area to store live-out data. See Fig.2, and note the ISA-visible register file. Clearly, instructions of a particular group (for instance, those of Fig.5) will be writing to registers within the file. These registers may in turn be used by a subsequent group of instructions.

42. Referring to claim 6, Ranganathan in view of Nair has taught a logic circuit as described in claim 5. Ranganathan has further taught a fifth storage area coupled to the control flow logic, the fifth storage area to map live-in and live-out data. See Fig.3, and note the register queues. The register queues, according to section 3.4, implement register-renaming, which is known to be the mapping of live-in and live-out data.

43. Referring to claim 7, Ranganathan in view of Nair has taught a logic circuit as described in claim 6. Ranganathan has further taught that each of the storage areas are in at least one memory device. This is inherent because any component which includes a storage area is a memory device.

44. Referring to claim 8, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught that the trace descriptor includes aggregate live-in data for a plurality of dependency descriptors in the trace description. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions depend on data which was produced prior to execution of the given trace. This data is live-in data and is associated with registers, for instance. Clearly, the trace descriptor will include this live-in data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note

that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-in data is for a plurality of dependency descriptors.

45. Referring to claim 9, Ranganathan in view of Nair has taught a logic circuit as described in claim 1. Ranganathan has further taught that the trace descriptor includes aggregate live-out data for a plurality of dependency descriptors in the trace description. Again, see Fig.5, and note that a trace includes a number of instructions. These instructions produce data which will be depended upon in the future. This data is live-out data and is associated with registers, for instance. Clearly, the trace descriptor will include this live-out data as it is needed to determine dependencies within the trace so that the trace may be split into PEW groups. Note that a plurality of dependency data (dependency descriptors) must exist in order to specify dependencies within a group of instructions. Therefore, the live-out data is for a plurality of dependency descriptors.

46. Claims 23-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Batten, as applied above, and further in view of Arimilli et al., U.S. Patent No. 6,427,204 (as applied in the previous Office Action and herein referred to as Arimilli).

47. Referring to claim 23, Batten has taught a method as described in claim 22. Batten has not taught that the fetching of a set of instructions is completed just in time for execution. However, Arimilli has taught such a concept. See column 3, lines 1-17. Note that Arimilli has taught that this is a more efficient way of fetching because instructions are only delivered when they are actually needed and pipeline bubbles are prevented. Consequently, it would have been

obvious to one of ordinary skill in the art at the time of the invention to modify Batten such that instructions are fetched just-in-time, as taught by Arimilli.

48. Referring to claim 24, Batten in view of Arimilli has taught a method as described in claim 23. Although Batten has not taught that the instructions are out of order, Arimilli has taught such a concept. See column 1, line 61, to column 2, line 6. Note that the use of resources and efficiency are maximized with out-of-order execution. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten to include instructions that are out-of-order, as taught by Arimilli.

49. Claims 25-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Batten in view of Arimilli, as applied above, in view of Witt et al., U.S. Patent No. 6,018,798 (as applied in the previous Office Action and herein referred to as Witt).

50. Referring to claim 25, Batten in view of Arimilli has taught a method as described in claim 24. Batten in view of Arimilli has not explicitly taught updating the architectural state using the data in the first storage area. However, Witt has taught the concept of having a speculative register file (future file 88, Fig.3) and an actual register file (Fig.3, component 102). The speculative register file holds the most current state of the machine (values determined via speculative execution) and by doing this, instructions may be executed speculatively. Once it is determined that instructions are no longer speculative, the speculative results are made architectural results by writing them to the actual register file. See column 12, line 66, to column 13, line 45. This is a known concept in the art. In essence, this scheme allows for speculative execution which is a method of executing instructions before it is known that they should

execute (they are predicted to execute). This maximizes efficiency if they indeed were to execute (predicted correctly). As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten in view of Arimilli such that the architectural state is updated using the data in the speculative storage.

51. Referring to claim 26, Batten in view of Arimilli and further in view of Witt has taught a method as described in claim 25. Witt has further taught recovering an earlier architectural state after a misprediction using the data in the first storage area. See column 18, lines 54-67, and note that after a misprediction, a previous state is achieved by copying actual values into the future file (so that the speculative values are correct). Consequently, by using this newly written data, the system recovers an earlier architectural state.

52. Claim 27 is rejected under 35 U.S.C. 103(a) as being unpatentable over Batten, as applied above, and further in view of Rotenberg et al., "A Trace Cache Microarchitecture and Evaluation," 1998 (as applied in the previous Office Action and herein referred to as Rotenberg).

53. Referring to claim 27, Batten has taught a method as described in claim 20. Batten has not taught that the selecting involves predicting the next trace descriptor to process. However, Rotenberg has taught such a concept. See page 3, and note the first paragraph under Figure 2. Rotenberg has taught that predicting sequences of traces for the implicit achievement of high branch prediction throughput. As a result, in order to increase branch prediction throughput, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Batten to include trace prediction as taught by Rotenberg.

Response to Arguments

54. Applicant's arguments filed on December 28, 2004, have been fully considered but they are not persuasive.

55. Applicant argues the novelty/rejection of claim 20 on page 13 of the remarks, in substance that:

> "In regard to independent claim 20, this claim includes the elements of "fetching the set of instructions at a location indicated by a dependency descriptor." Similar to the elements of independent claim 1 and the discussion related thereto, Batten does not teach a dependency descriptor that includes a location of a set of instructions. Rather, the dtype field of Batten indicates only data hazards found within the set of instructions associated with a ccdd instruction. Thus, Batten does not teach each of the elements of independent claim 20. Accordingly, reconsideration and the withdrawal of the anticipation rejection of independent claim 20 are requested."

56. These arguments are not found persuasive for the following reasons:

a) In response to applicant's amendments, the dependency descriptor of Batten is interpreted as comprising at least the dtype field and the numInstr field (Fig.7). The numInstr field specifies the number of subsequent instructions associated with the dependencies. This field indicates a location of the sequence because it says "the next X instructions are associated with these dependencies". The location is the next X addresses after the ccdd instruction.

Conclusion

57. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO

MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

DJH
David J. Huisman
February 28, 2005

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100